

Functions

Announcements

Expressions

Types of expressions

An expression describes a computation and evaluates to a value

$$18 + 69$$

$$\frac{6}{23}$$

$$\sin \pi$$

$$\log_2 1024$$

$$2^{100}$$

$$f(x)$$

$$\sqrt{3493161}$$

$$7 \bmod 2$$

$$\sum_{i=1}^{100} i$$

$$\lim_{x \rightarrow \infty} \frac{1}{x}$$

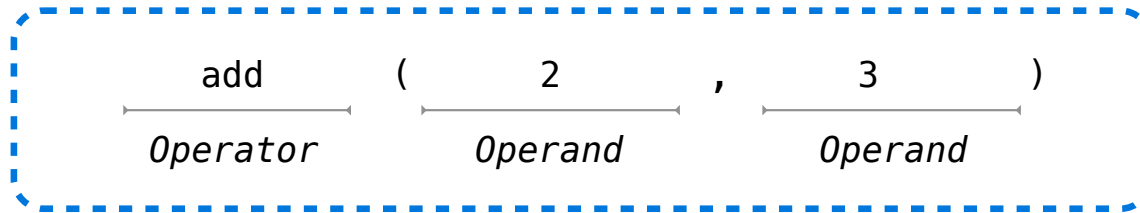
$$|-1869|$$

$$\binom{69}{18}$$

Call Expressions in Python

All expressions can use function call notation
(Demo)

Anatomy of a Call Expression



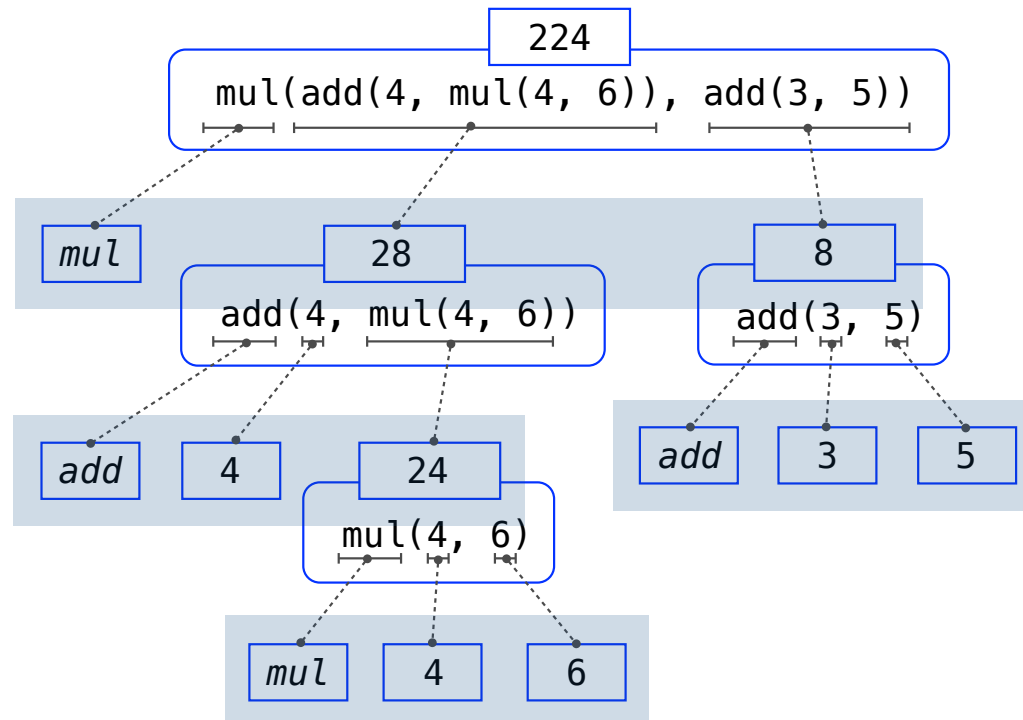
Operators and operands are also expressions

So they evaluate to values

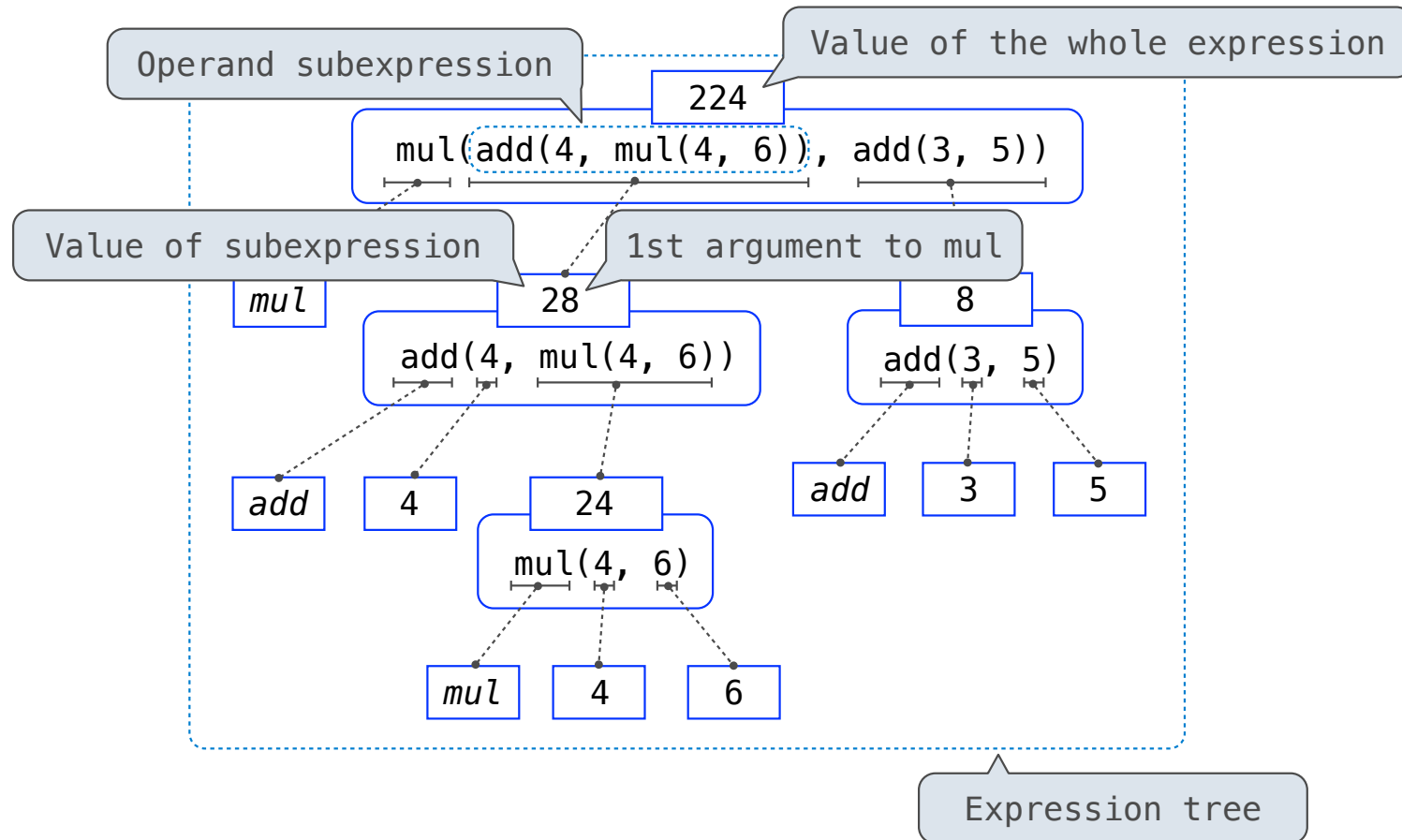
Evaluation procedure for call expressions:

1. Evaluate the operator and then the operand subexpressions
2. **Apply** the **function** that is the value of the operator to the **arguments** that are the values of the operands

Evaluating Nested Expressions



Evaluating Nested Expressions

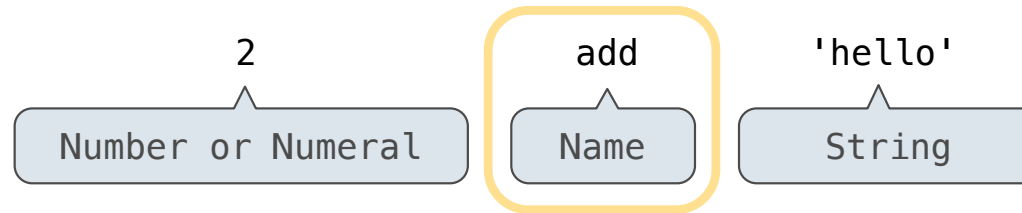


Names, Assignment, and User-Defined Functions

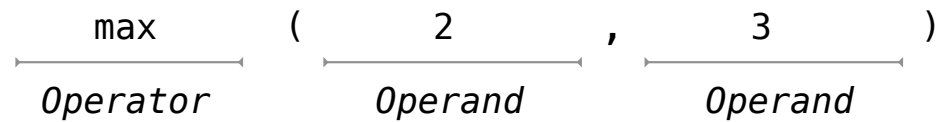
(Demo)

Types of Expressions

Primitive expressions:



Call expressions:



An operand can also be a call expression

`max(min(pow(3, 5), -4), min(1, -2))`

Discussion Question 1

What is the value of the final expression in this sequence?

```
>>> f = min
```

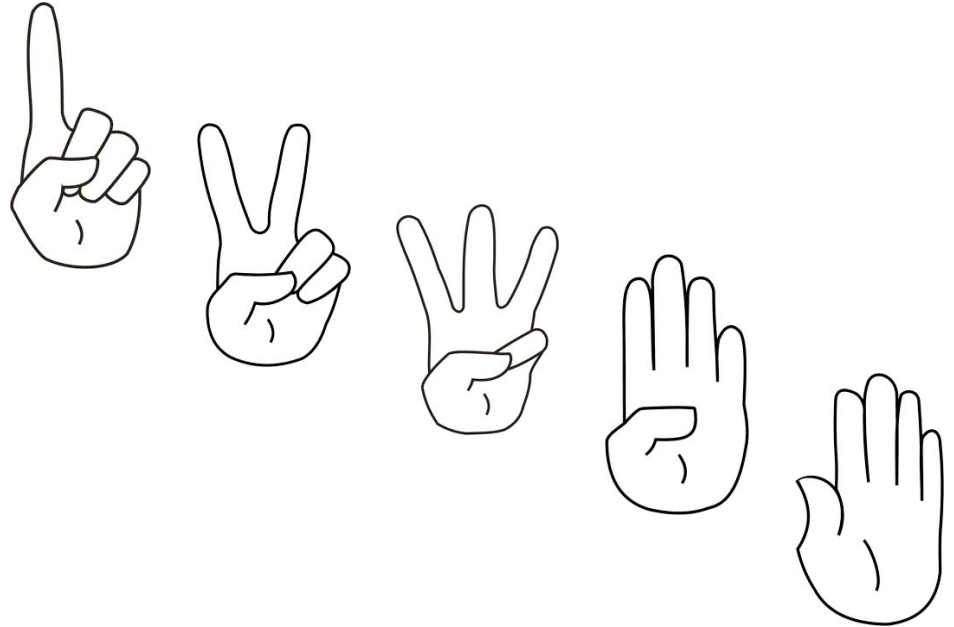
```
>>> f = max
```

```
>>> g, h = min, max
```

```
>>> max = g
```

```
>>> max(f(2, g(h(1, 5), 3)), 4)
```

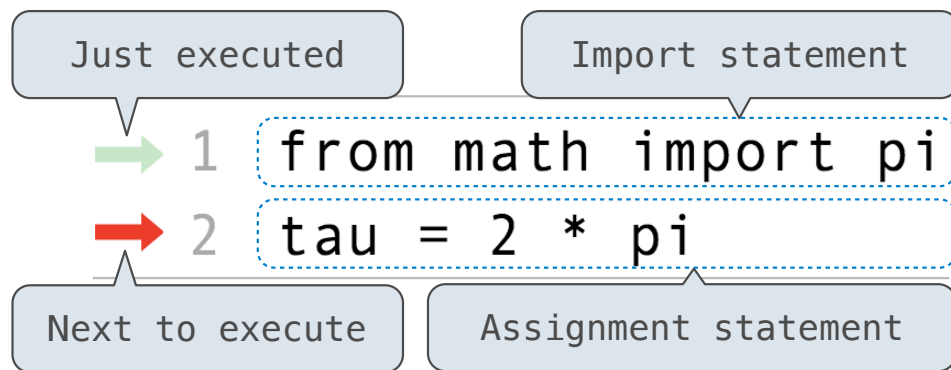
???



Environment Diagrams

Environment Diagrams

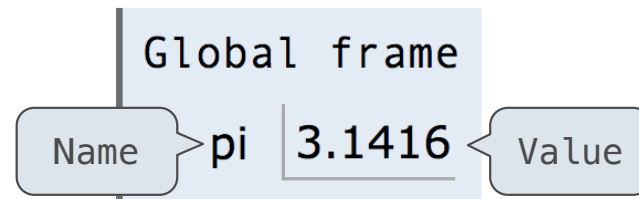
Environment diagrams visualize the interpreter's process.



Code (left):

Statements and expressions

Arrows indicate evaluation order



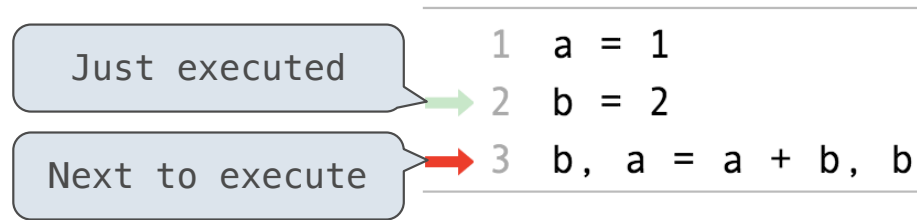
Frames (right):

Each name is bound to a value

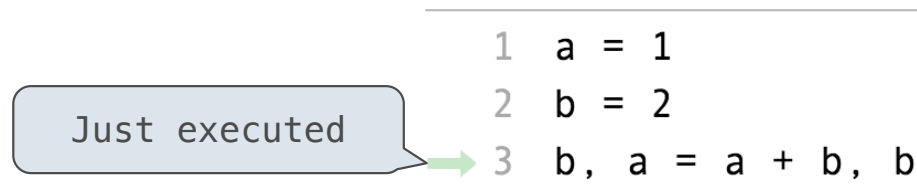
Within a frame, a name cannot be repeated

(Demo)

Assignment Statements



```
Global frame
a | 1
b | 2
```



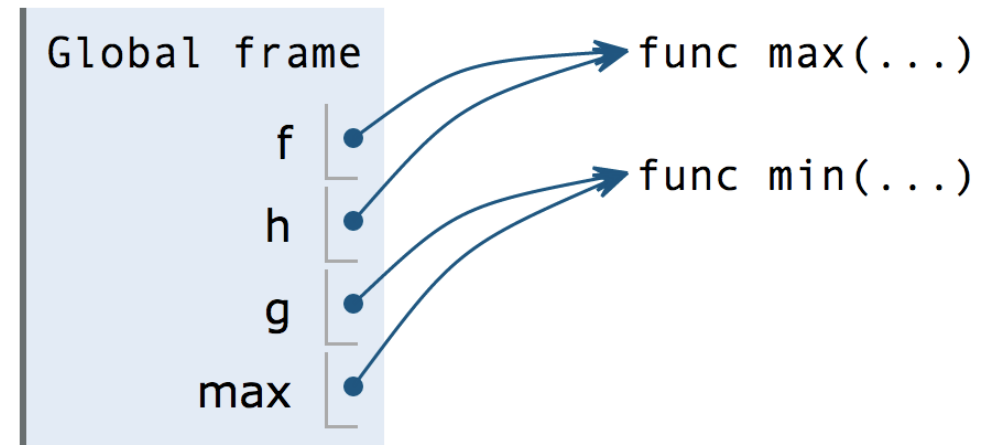
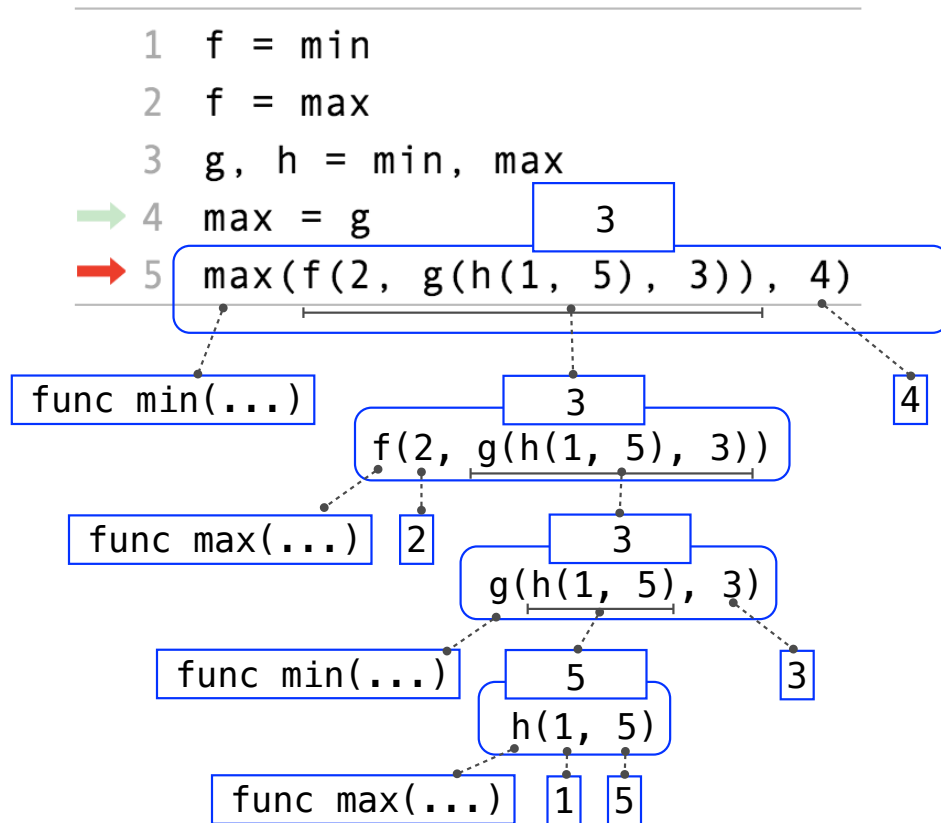
```
Global frame
a | 2
b | 3
```

Execution rule for assignment statements:

1. Evaluate all expressions to the right of = from left to right.
2. Bind all names to the left of = to those resulting values in the current frame.

Discussion Question 1 Solution

(Demo)



3

Defining Functions

Defining Functions

Assignment is a simple means of abstraction: binds names to values

Function definition is a more powerful means of abstraction: binds names to expressions

Function **signature** indicates how many arguments a function takes

```
>>> def <name>(<formal parameters>):  
    return <return expression>
```

Function **body** defines the computation performed when the function is applied

Execution procedure for def statements:

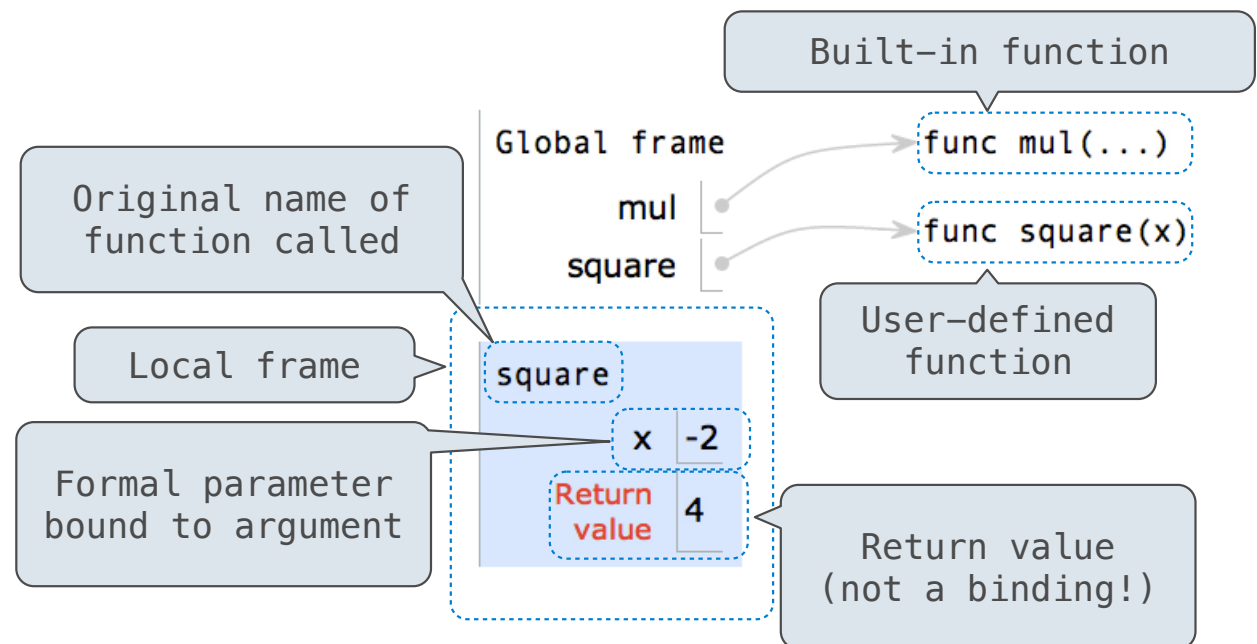
1. Create a function with signature `<name>(<formal parameters>)`
2. Set the body of that function to be everything indented after the first line
3. Bind `<name>` to that function in the current frame

Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(-2)
```



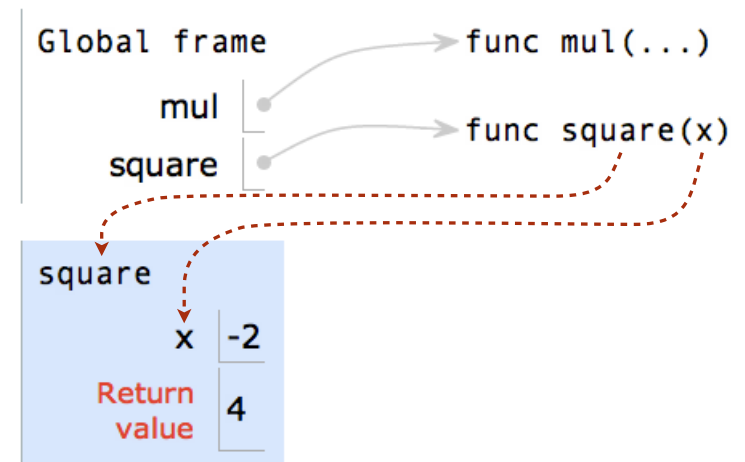
Calling User-Defined Functions

Procedure for calling/applying user-defined functions (version 1):

1. Add a local frame, forming a new environment
2. Bind the function's formal parameters to its arguments in that frame
3. Execute the body of the function in that new environment

```
1 from operator import mul
2 def square(x):
3     return mul(x, x)
4 square(-2)
```

A function's signature has all the information needed to create a local frame



Looking Up Names In Environments

Every expression is evaluated in the context of **an environment**.

So far, the current environment is either:

- The global frame alone, or
- A local frame, followed by the global frame.

Most important two things I'll say all day:

An environment is a sequence of frames.

A name evaluates to the value bound to that name **in the earliest frame** of the current environment in which that name is found.

E.g., to look up some name in the body of the square function:

- Look for that name in the **local frame**.
- If not found, look for it in the **global frame**.
(Built-in names like “max” are in the global frame too, but we don't draw them in environment diagrams.)

(Demo)